

1 SHARED VIRTUAL DESKTOP COLLABORATIVE
2 APPLICATION SYSTEM
3
4

5 Inventor:
6 Daniel W. Wright
7
8

9 Background of the Invention

10 1. Field of the Invention:

11 The present invention is generally related to computer systems that
12 provide for the inter-networked simultaneous sharing of information and, in
13 particular, to a collaborative computer application system that provides for a
14 virtual shared application space.
15

16 2. Description of the Related Art:

17 With the recent expansion in the variety of information technologies
18 and the distribution of information among network interconnected, or inter-
19 networked computer systems and users, a need has arisen to coordinate the
20 exchange and development of information by users typically at separate and
21 potentially heterogeneous computers systems. In many of these instances, the
22 information that needs to be shared or created requires the collaborative or
23 effectively simultaneous use of some particular application. In many of these
24 instances, the application has been originally designed and implemented to utilize
25 a virtual display window created within and managed by a graphical user

1 interface (GUI) based operating system. Many implementations of such operating
2 systems are well known and include the Apple Macintosh System 7 operating
3 system, the MicroSoft MS-Windows 3.1, MS-Windows 95, MS-Windows NT, and
4 the XWindows system, originally developed at MIT and used in many Unix based
5 operating systems, including SunOS and Solaris.

6
7 The virtualization of the display in the form of a window permits some
8 small conventional degree of flexibility in controlling where underlying data and
9 programs are stored, whether an application program is locally or remotely
10 executed and whether the display window or windows utilized by the particular
11 application are locally or remotely displayed. Although these degrees of flexibility
12 are conventionally available, collaborative use of an application program to
13 interactively or simultaneously exchange and create information is not effectively
14 supported. Conventional application programs remain by and large single user
15 tools. As such, these applications co-exists in a networked environment
16 constrained to sharing data through low level record and file locked access to
17 shared data on network accessible storage devices.

18
19 Conventional collaborative application programs have been designed and
20 implemented in an effort to support a greater degree of concurrent interactivity.
21 Most conventional collaborative applications are highly proprietary in that the
22 manner and nature of the permitted collaboration is strictly controlled by the
23 particular application. Collaborative interaction is confined to the functions of
24 that application. One manner by which such applications operate is through the
25 establishment of a background server accessible by way of a network connection

1 from each host that will participate in a collaborative session. On each
2 participant host, a user executes an identical copy of the collaborative application.
3 The application itself is responsible for initiating or joining a collaborative session
4 by registering with the background server through a proprietary protocol, though
5 typically overlaid on a conventional protocol such as TCP/IP. Thereafter, for the
6 duration of the collaborative session, the application is responsible for duplicating
7 all collaborative user input to the application and forwarding the copied input to
8 the background server. In turn, the background server is responsible for re-
9 distribution of all collaborative input from each participating host to all other
10 participating hosts. Thus, each of the collaborative applications operate from the
11 cumulative set of collaborative user input. Consequently, each application is
12 expected to operate and present information in a synchronized manner.

13
14 Proprietary collaborative application programs, while generally functional
15 for their intended purposes, are of limited collaborative value because the
16 collaborative function supported is specifically limited to that of the particular
17 application. Often, the particular requirements of a specific collaborative
18 application program, in order to function as intended, may bar the use of other
19 collaborative applications at the same time by the participating hosts.
20 Furthermore, a high degree of administrative overhead, if not also computer
21 processing overhead, is often required to support collaborative application
22 programs. These costs are additive to the processing and administration
23 requirements of the underlying operating system and networking support required
24 by the application. Thus, collaborative applications have been effective most

1 typically in situations where specialized use and particular functionality have been
2 required.

3
4 An alternative to the use of proprietary collaborative application programs
5 is a technology known as screen sharing. This technology provides for the
6 information displayed on the display of a primary or host computer system to be
7 projected across a network to another, or guest computer system. In general,
8 screen sharing is implemented through establishment of a logical tap at the
9 display device driver level of the host computer system. All display data is
10 duplicated by way of this tap and passed through the network to the guest. On
11 the guest system, the screen sharing application executes a logically inverse data
12 tap to display the monitored screen data on the display of the guest.

13
14 Screen sharing therefore operates largely independent of the particular
15 applications being executed on the host in addition to the screen sharing
16 application. Screen sharing allows application independence to the point that
17 collaborative sharing of well behaved though otherwise ordinary applications,
18 including applications that otherwise could not be executed on a particular guest
19 system, can be made subject to the collaboration.

20
21 Unfortunately, conventional screen sharing effectively precludes the private
22 co-execution of other applications on both the host and all guest systems during
23 the collaboration. All user input on both the host and guest systems is provided
24 to the host executed applications. Thus, the entire function of the host and guest
25 appears synchronized and limited to the collaboration. Consequently, screen

1 sharing is predominately used to allow remote systems to monitor the display
2 oriented aspects of the execution of applications on a single host system.

3
4 Another form of computer based collaboration is known as window
5 sharing. In collaborative window sharing, the host system executes an otherwise
6 conventional application within a window established and managed by a
7 proprietary window sharing collaborative application. As with screen sharing, the
8 principal operative feature of window sharing is a tapped duplication of the
9 window display data for transfer to one or more guest systems participating in the
10 window sharing collaboration. Each of these guests also executes the window
11 sharing application, though configured to receive and display the tapped data in
12 a similarly configured window on the guest system. Since a single application is
13 being executed within the logical confines of the shared window, guest input data
14 can also be tapped and provided to the local host at least in a two system
15 collaboration session. If more than two systems are to participate share input
16 data in a collaboration, a significantly more complex registration and input server
17 system is generally required.

18
19 The window sharing technology is further constrained in general by the
20 limitation that only a single application can be collaboratively shared within a
21 single window. A collaboration session could be realized through a single window
22 by execution of a succession of applications. To avoid the need to stop and restart
23 applications, multiple windows may be supported by the window sharing
24 application. As expected, the co-execution of applications in respective windows
25 will contend with one another for system resources. However, the execution

1 behavior of the shared applications may be unusual due to the potential of
2 unexpected inputs. A first application may be executed on the host computer
3 system and shared with one or more guests. If the user of that system changes the
4 local input focus to a second window, corresponding to either a private, locally
5 executed application or to another shared application window, the focus event will
6 typically suspend execution of the first application until a focus event within the
7 application's window returns execution focus to the first application. While
8 suspended, the shared application will refuse all input except for a focus event,
9 such as a mouse click within the shared window. Thus, all collaborative guests
10 are suddenly and unexpectedly stopped in the midst of their collaboration.

11
12 A shared application focus event may also be generated on any of the
13 guest systems. Consequently, a shared window may be suddenly and
14 unexpectedly raised to active execution on the host by a guest focus event. That
15 is, each time any collaborator at the host or any guest system introduces a focus
16 event into a guest shared window, focus and execution will immediately switch to
17 the shared application on the host and all guests. If multiple shared applications
18 are being co-executed on the host, the contention for focus will be substantial.
19 Thus, users at the host and guest executing one or more shared applications may
20 be sharply limited if not barred as a practical matter from co-execution of other
21 applications, shared or private. This characteristic of window sharing is, in
22 general, poorly received by the users of such applications.

23
24 A combination window and screen sharing technology is also known. This
25 technology provides for the sharing of the full screen on the host. The shared

1 screen is, however, displayed in a window on a guest system. By sharing the full
2 host screen, multiple host executed applications can be shared within a single
3 collaboration session. Input events are mapped on the guest system to be window
4 frame relative for events within the shared window. Consequently, a reasonably
5 operative desk top is represented on the guest in the shared window.
6

7 However, the combined window and screen sharing technology inherits
8 most if not all of the disadvantages of the individual screen sharing and window
9 sharing technologies. On the host, no private applications can be executed since
10 the entire screen is shared. While focus events from a guest outside of the guests'
11 shared window will not be shared with the host, all host focus events will generate
12 shared window focus events on a guest. Thus, any activity on the host or through
13 the host from potentially other guests will raise the shared windows of all guests.
14 The ability to execute private applications on the guest systems is therefore largely
15 defeated.
16
17

18 Summary of the Invention

19 Thus, a general purpose of the present invention is to provide a virtual
20 shared application space that enables collaborative information exchange.
21

22 This is achieved in the present invention by provision of a computer system
23 that executes first and second sets of application programs. The computer system
24 includes a processor that includes an input device and an output device and an
25 operating system executed in support of the execution of programs. The

1 operating system includes a graphical user interface coupleable through an
2 output driver to the output device and an input interface including an input queue
3 coupleable through an input driver to the input device. The operating system also
4 includes a first list of a first set of application programs executable by the
5 processor and a second list of application program windows corresponding to the
6 first set of application programs. An environment manager program is also
7 executed by the processor. The environment manager includes a third list of a
8 second set of application programs and a fourth list of application program
9 windows corresponding to the second list of application programs. Execution of
10 the environment manager provides for selectively swapping with the operating
11 system the first and third lists and the second and fourth lists to switch between the
12 execution of the first and second sets of application programs.

13
14 Thus, an advantage of the present invention is that a functional virtual
15 application space is created.

16
17 Another advantage of the present invention is that a consistent, complete
18 user environment is created.

19
20 A further advantage of the present invention is that the relationship
21 between the shared virtual application space and the non-shared application
22 spaces of inter-networked computer systems mutually inter-operate in well defined
23 and consistent manner.

1 Yet another advantage of the present invention is that the shared virtual
2 application space maintains an independence from the specific devices that
3 provide input events, provide for display outputs and establish the inter-
4 networking between collaboratively operated computer systems.

5
6 A still further advantage of the present invention is that a collaborative
7 environment manager can be provided in the form of an application, device
8 driver, system library or combination thereof to establish a collaborative shared
9 virtual application space fully consistent with the normal operation of a native
10 operating system executed by a given computer.

11
12 Still another advantage of the present invention is that the system
13 architecture employed to establish the shared collaborative application space is
14 consistent with the implementation of generic graphical user interfaces, thereby
15 permitting collaborative operation on heterogeneous systems.

16
17 Yet still another advantage of the present invention is that the inter-
18 networking communication between collaborative systems is optimized and readily
19 inclusive of incorporating fully collaborative participation by multiple computer
20 systems in the shared application space.

21
22
23 Brief Description of the Drawings
24

1 These and other advantages and features of the present invention will
2 become better understood upon consideration of the following detailed
3 description of the invention when considered in connection of the accompanying
4 drawings, in which like reference numerals designate like parts throughout the
5 figures thereof, and wherein:

6
7 Figure 1a is a schematic block diagram of a computer system suitable for
8 use with the present invention;

9
10 Figure 1b is schematic block diagram of an event driven operating system
11 supporting a graphical user interface consistent with the utilization of the present
12 invention;

13
14 Figure 2 is a simplified schematic block diagram of the components of a
15 collaborative application environment manager consistent with the present
16 invention;

17
18 Figure 3 provides a simplified block diagram illustrating the establishment
19 of overt and covert operating environment and shared participation in both by the
20 environment manager application of the present invention;

21
22 Figure 4 provides a schematic block diagram illustrating the
23 implementation and operation of the present invention in establishing the covert
24 display environment;

1 Figures 5a and 5b provide graphical representations of the relationship
2 between the overt and covert windows established by the environment manager
3 of the present invention;
4

5 Figure 6 provides a flow diagram illustrating the relevant coercion of the
6 event manager of an event driven operating system consistent with the present
7 invention;
8

9 Figure 7 provides a flow diagram illustrating the modified event handling
10 loop for the overt environment and a single covert environment in accordance
11 with a preferred embodiment of the present invention;
12

13 Figure 8 provides a detailed flow diagram of the input event subsystem of
14 an event driven operating system consistent with the present invention; and
15

16 Figure 9 provides a flow diagram illustrating the modified input handling
17 routine implemented in accordance with the preferred embodiment of the present
18 invention.
19

20 Detailed Description of the Invention

21

22 A computer system 10, as shown in Figure 1a, is suitable for use in
23 execution of an event driven operating system supporting a graphical user
24 interface appropriate for use in connection with the present invention. The
25 computer system 10 includes a micro-processor 12 for executing an operating

1 system and one or more application programs stored in a memory system 14 and
2 accessible by way of an inter-connecting data and control bus 16. Display
3 information is processed through a display controller 18 for rendering on a
4 display 20. Inter-networking communications data is processed through a
5 network interface controller 22 to a network 24. A minimum, though presently
6 preferred, implementation of the interface controller 22 is as a standard serial
7 port providing an inter-networking path over a point-to-point serial path 24. The
8 preferred serial connection, along with the relative simplicity of selecting particular
9 guests for collaboration, is preferred as fully sufficient for collaborative sessions
10 established with the use of digital simultaneous voice and data (DSVD) modems.
11 The interface controller 22 may also be an ethernet or similar local area network
12 (LAN) adapter providing for connectivity to a conventional ethernet network 24.
13 Finally, an input interface controller 26 is provided to receive input data typically
14 from a mouse 28 and a keyboard 30. The interface controller 26, as in many
15 conventional implementations, includes a serial port and a dedicated keyboard
16 controller.

17
18 Conventional computer systems generally sufficient to provide the basic
19 system requirements of the computer system 10 include personal computers
20 based on the IBM AT architecture and derivatives thereof, Apple Macintosh and
21 PowerPC computers, and Sun SparcStations, among others.

22
23 Referring now to Figure 1b, a representative diagram of an event driven
24 operating system 40 supporting a graphical user interface (GUI) is shown. The
25 operating system 40 provides both input and output control services for a number

1 of applications 42_1-42_n that are executed by the computer system 10 with the
2 support of the operating system 40. Characteristic of the operating system 40 is
3 that input data is received and stored by the operating system as discrete events
4 in an input queue 56. As events, each input datum is stored in the input queue
5 with an effective identification of the data source and the data destination. The
6 data destination information may be subsequently utilized by the operating system
7 40 to release the data to the appropriate application 42_1-42_n for which the data
8 is intended.

9
10 An input handler routine 58 is typically provided as a part of the operating
11 system 40 to manage the receipt of input data and to appropriately store the data
12 in the input queue 56. The input handler executes typically in response to the
13 receipt of a hardware interrupt generated coincident with the receipt of input data.
14 The input handler 58, in turn, obtains the input data, creates an identifier as to the
15 source and destination of the data, and stores both as an entry in the input queue
16 56. Each entry in the input queue 56 is, in the preferred embodiment, a list
17 element in a linked list data storage structure or equivalent.

18
19 The operating system 40 also preferably contains a data structure 60 that
20 provides slotted entries for each of the applications 42_1-42_n that have been
21 launched for execution with the support of the operating system 40. Each of the
22 slotted entries in the application list 60 contains identifying information locating
23 each of the applications 42_1-42_n within the address space of the operating system
24 40. The slotted entries also store other information as may be required by the
25 operating system to establish and maintain the memory space and effective

1 execution state of each application. A slotted entry in the application list 60 is
2 typically allocated or initialized upon launch of a new application. As the
3 application is loaded into memory within the address space of the operating
4 system 40, memory is allocated based on the memory space requirements of the
5 application. Appropriate identifiers for this information are entered into the
6 slotted entry for the application in the application list 60. Once loaded, state
7 information also typically stored in the slotted entry is initialized to identify the
8 application as ready to run.

9
10 When an application is terminated, the corresponding memory space
11 identifiers are utilized to release the memory occupied by the application for
12 subsequent use by the operating system 40. The entry in the application list 60
13 is either deleted or marked as unused.

14
15 Typically, a dynamically allocated hierarchical or tree structured window
16 list is utilized by the operating system 40 in management of the windows W_1 - W_n
17 that are opened upon request by the applications 42_1 - 42_n . The windows W_1 - W_n
18 are, in the preferred embodiment, allocated within the address space of the
19 respective applications 42_1 - 42_n and initialized with a logical representation of a
20 window display. The window memory is allocated in response to operating system
21 calls made by the executing application through the application program interface
22 (API) of the operating systems 40. Other calls can also be made by the
23 application to specify various window decorations and to establish callbacks to the
24 application upon user manipulation of the associated window decorations

1 With each call to create a window, the operating system 40 adds a list entry
2 to the tree structured window list. The logical root of this window list is the desktop
3 or root window of the operating system 40. Typically, a single word of data,
4 utilized as a pointer to the root window list entry of the tree structured list, is stored
5 in a root window pointer location 62. The root pointer is initialized to point to a
6 root window list entry. The root window entry, like all other tree structured window
7 list entries, typically includes structure links sufficient to allow a linked list type
8 traversal of the tree structured window list by the operating system 40. Traversals
9 of the window list are performed by the operating system 40 to determine window
10 boundaries and visibility in connection with display update events, for example.

11
12 Each entry in the window list also provides storage for a pointer reference
13 to the base of a window memory space W_1-W_n existent within a corresponding
14 application 42_1-42_n . Thus, as each application window W_1-W_n is created,
15 additional window list entries are allocated and linked below the root window to
16 form the tree structured window list. By the ordered linkage of the entries, a
17 hierarchical relation is established representing the desired display window
18 hierarchy over the root or desktop window.

19
20 Another data word is separately stored by the operating system 40 in an
21 active window pointer location 64. The pointer reference stored in the location 64
22 may be a pointer into the application entry list 60 or directly to a particular
23 application in memory. This referenced application is identified as the currently
24 executing foreground application. As such, the application is also the one
25 application that currently has the input focus of the operating system 40. Any

1 other applications may continue to effectively execute in the background as
2 supported by the operating system 40, at least until user input is provided. In
3 general, the application that is executing within the window with input focus
4 receives normal keyboard and mouse input data. Events that are specified to
5 change the input focus, such as a mouse click in another window, may shift the
6 input focus to another window and therefore direct input to another application.
7 With each change in input focus, the pointer stored in the active window pointer
8 location 64 is correspondingly updated to reference the new active foreground
9 executing application.

10
11 Finally, the operating system 42 preferably provides for the storage of a
12 drawing routine table 66. This table 66 is utilized as a jump table permitting
13 integration of a device specific display driver with the virtual display drawing
14 operations internally supported by the operating system 40. In the preferred
15 embodiment, an executing application 42₁-42_n may issue an series of virtualized
16 API drawing command calls to the operating system 40. These calls are generally
17 intended to direct the operating system 40 to make corresponding modifications
18 to the contents of a the window of the requesting application. Where the display
19 area of other windows may be affected by the requested window modifications,
20 as may be determined from a traversal of the window list, update events are
21 queued for the applications owning the affected windows.

22
23 The operating system 40 processes each of the virtualized drawing
24 commands into a series of one or more well-defined atomic drawing operations.
25 These atomic operations are implemented by sub-routines within a device driver

1 46 and are intended to incorporate the hardware specific drawing and display
2 functions of the display hardware 48. In order to integrate these atomic drawing
3 routines into the function of the operating system 40, the drawing routine table 66
4 is used to store execution jump pointers to the individual display controller specific
5 sub-routines 46. The table 66 is initialized with these jump addresses upon the
6 original loading and initialization of the display driver 46 by the operating system
7 40. Thereafter, the atomic drawing commands are accessed through low level
8 API calls by the operating system 40 to physically reflect the logical state of the
9 data stored within the window address space W_1 - W_n of the originally requesting
10 application 42_1 - 42_n .

11
12 The display driver 46 drawing commands perform corresponding drawing
13 operations on the provided display data. The resulting bit map representation of
14 the display data is written to the video buffer space present on the display
15 hardware 48. The display data is provided by the operating system 40 as needed
16 to maintain the currency of the logically visible portions of the windows W_1 - W_n .
17 That is, applications will process display update events or other events that
18 ultimately require the updating of a portion of the display to reflect an exposure
19 of a portion of a window or to change the display contents of a window. In either
20 instance, the operating system 40 is called upon by the applications 42_1 - 42_n to
21 cause operating system 40 to issue a series of atomic drawing commands. The
22 commands issued and data associated with the commands is determined in part
23 by a partial or complete traversal of the tree structured window list whose root is
24 currently pointed to by the tree root pointer 62. As the tree structure is traversed,
25 the visible portion of the windows W_1 - W_n are determined by the operating system

1 40 and transferred by execution of atomic drawing commands to the video buffer
2 memory on the display hardware 48.

3
4 Finally, for purposes of the present invention, a communication driver 50
5 is preferably provided in conventional connection to the operating system 40. The
6 communication driver 50 and associated communication hardware 52 may be
7 implemented as a simple serial data sub-system or as a more complicated local
8 area network sub-system. In the preferred embodiment of the present invention,
9 the communication driver 50 is implemented as a serial port driver routine and
10 the communication hardware 52 is a conventional serial port. Thus, the external
11 data connection 54 represents a point-to-point serial link connecting the system
12 38 to a second computer system 10' also executing a functionally equivalent
13 system 38'. The operating system 40 may be called by an application 42₁-42_n to
14 establish a logical connection between the application and the serial port drive
15 routine. This data received by the serial port communication hardware 52 is
16 routed by the operating system 40 to the application 42₁-42_n typically through a
17 data queue dedicated to the port.

18
19 Alternately, the communication driver 50 may include a PC-TCP/IP stack
20 with an appropriate data link layer for managing the operation of an ethernet
21 network interface adapter 52 or token ring network interface adapter. The
22 network 54 can thus provide for the interconnection of one or more other
23 computer systems 10' each executing a local copy of the system 38'. Again, the
24 operating system 40 provides a logical relation between the application and the
25 communication hardware 52, further augmented by the use of internet protocol

(IP) addresses or the like. Thus, received network data is routed through a dedicated network data queue to the correct destination application 42₁-42_n.

The system 10' may also be constructed with a quite different hardware architecture and execute a different operating system 40' so long as the systems 38, 38' are logically consistent in their implementation of present invention.

Referring now to Figure 2, an environment manager application 42₃ appropriate for execution within an environment supported by the event driven operating system 40 is shown. The environment manager application 42₃ incorporates a number of data structures and support sub-routines appropriate to implement a preferred embodiment of the present invention. The environment manager application 42₃ includes an alternate or covert application list 70 equivalent in logical structure to application list 60, an alternate storage location 72 for storage of a root window pointer to an alternate tree structured window list, and in an alternate active window storage location 74 for storage of an alternate active window pointer. The environment manager application 42₃ further includes an alternate input queue 76 and a modified input queue handler routine 78. Finally, the application 42₃ includes an alternate drawing routine jump table 80 and a covert environment device driver 82 containing an alternate set of well-defined atomic drawing routines. The entry points for these alternate atomic drawing routines are stored in the conventional sequence in the alternate jump table 80.

The purpose of the alternate data structures and routines provided as part of the environment manager application 42₃ is to permit, in general terms, a swapping of the original or overt structures and routines with the alternate or covert structures and routines on a dynamic basis to switch the global context of the operating system 40 between a overt environment involving the execution of one set of applications and a covert environment involving the execution of a second set of applications. The overt environment applications execute within the normal environment of the operating system 40 as conventional private or entirely locally executed applications on the host 10. The covert environment applications are also executed on the host 10 as local applications, but subject to environmental modifications that enable the sharing of input events and atomic display commands among any number of collaborative guests 10' and the host 10. That is, in combination with the swapping in of the covert data structures and routines, local or server input events as well as remote or guest input events from the collaborative guests 10' are collected and provided as input events to the applications in the covert environment. At the same time, all atomic drawing commands are directed to the covert display device driver 82 which is specifically modified to utilize the memory space of the environment manager application window W₃ as a display data buffer for the covert environment display. The root display space of the covert environment is thus presented within the overt environment as the displayed contents of environment manager application window W₃.

The atomic drawing commands are also forwarded by operation of the covert display driver through the environment manager 42 and operating system

1 40 to the collaborative guests 10' to correspondingly update a client collaborative
2 application window W_3' . That is, the environment manager applications 42_3 of
3 the collaborative guests 10', upon establishing a serial or network connection with
4 the host 10, configures a client collaborative application window W_3' . All input
5 events in this window W_3' are forwarded through the environment managers $42_3'$
6 and operating system 40' over the network 24 to the host 10. All atomic drawing
7 commands forwarded by the host 10 are, in turn, applied by the environment
8 managers $42_3'$ to the operating system 40' to update the contents of the client
9 collaborative application window W_3' on the collaborative guests 10'.

10
11 As schematically indicated in Figure 3, a preferred embodiment of the
12 present invention particularly suited for execution in an MS Windows 3.1
13 operating system environment provides for an overt environment 90 and a covert
14 environment 92. The overt environment 90 may include any number of
15 executable applications OA_1 - OA_N . Likewise, the covert environment 92 may
16 include any number of covert executable applications CA_1 - CA_N . The total number
17 of overt and covert environment applications are naturally limited by the total
18 application space supported by the operating system 40.

19
20 As shown in Figure 3, for operating systems 40 that support cooperative
21 multi-tasking such as MS-Windows 3.1 and Macintosh System 7, an application
22 embodying the present invention is provided to execute in both the covert 90 and
23 overt 92 environments. Thus, at each opportunity for the shared application 42_3
24 to execute, a switch between the overt and covert environments may be

1 implemented. Preferably, a timer event is established to ensure that the shared
2 application 42₃ is periodically scheduled to execute.

3
4 In a generally preemptive operating system environments, such as provided
5 by MS-Windows '95, operating system task switch notification can be utilized to
6 directly execute a portion of the shared application 42₃ on each task switch. Thus,
7 with each task switch, a switch can be made between the overt and covert
8 environments as needed. In other operating systems, including MS-Windows NT
9 and Unix based operating systems that substantially support true preemptive
10 execution of applications, standard system services may be utilized to directly
11 invoke execution of the environment switching portion of the shared application
12 with each context switch implemented by execution of the system scheduler.

13
14 Figure 4 generally shows the preferred reconfiguration of the operating
15 system 40 when supporting application execution in the covert environment. The
16 alternate input queue 76 is effectively swapped for the overt input queue 56.
17 Since the input queues 56, 76 are data storage structures, the contents of the
18 queues 56, 76 are preferably exchange copied on entry and exit of the covert
19 environment. The modified input handler routine 78 is logically installed in place
20 of the input handler 58. Unlike the other alternate data structures and routines,
21 the modified input handler 78 remains resident for the duration of the execution
22 of the environment manager application 42₃. That is, the modified input handler
23 78 manages the input queues 56, 76 in both the overt and covert execution
24 environments for so long as a covert environment exists. This allows the modified
25 input handler 78 to recognize and handle input data for both the overt and covert

1 environments regardless of which environment is currently being supported by the
2 execution of the operation system 40. Also, in order to simplify the
3 implementation of the present invention, the input handler 58 is preserved and
4 used as a dedicated substantive of the new handler 76.

5
6 The display driver jump table 80 is exchange copied with the drawing jump
7 table 66 on entry into and exit from the covert environment. Consequently,
8 during execution within the covert environment, operating system calls are
9 directed through the drawing routine jump table 80 to the covert display driver 82.
10 All conventional applications executing within the covert environment are therefore
11 provided with the full services of what appears to those applications as a
12 conventional display driver.

13
14 In accordance with the present invention, the covert display driver 82 differs
15 from a conventional display driver in that the memory space of window W_3 of the
16 environment manager application 42₃ is utilized as a bit map based display data
17 buffer. That is, all window update events within the covert environment are
18 ultimately realized by the execution of atomic drawing commands executed
19 against the bit map memory space stored in the window W_3 . The contents of the
20 window W_3 , when transferred as a direct bit map in response to an update event
21 against the overt environment window W_3 , properly displays the hierarchial
22 arrangement of windows within the covert environment within the frame of the
23 window W_3 on the display 48.

1 The covert display driver 82 also differs in that the driver 82 provides for
2 a duplication and forwarding of all atomic display commands to the operating
3 system 40 effectively via the environment manager application 42₃.
4 Consequently, the commands are ultimately forwarded to the operating system 40
5 preferably includes a logical identification of each collaborative guest 10' that is
6 to receive the copied atomic display command.

7
8 Finally, the covert display driver 82 differs in that an update event is
9 generated and provided to the overt environment input queue 58 when the
10 content of the covert display window W_3 is changed. Thus, when the environment
11 manager application 42₃ swaps back to the overt environment, the relatively
12 conventional sequence of processing of outstanding update events by the
13 environment manager application 42₃ in the overt environment context will result
14 in an efficient updating of the overt environment window W_3 to the display 48.

15
16 The relationship between the overt and covert environments in terms of root
17 window displays is illustrated in Figure 5a and 5b. In Figure 5a, a root window
18 list entry represents an overt environment root window W_0 . A number of
19 heirarchially related overt application windows W_1 - W_{10} are further represented as
20 window list entries linked beneath the root display list entry. In general, each
21 application is responsible for updating the representation of its window to the root
22 display window. The need to update a particular window occurs in response to
23 the execution of the application that owns the window or the execution of another
24 application that results in, for example, the exposure of some part of the window.
25 Updates of a window by the owning application are typically implemented

1 through the generation of drawing events directed to the operating system.
2 Updates due to the execution of other applications typically result in the
3 generation of update events for the applications of the affected windows. Upon
4 execution, applications typically check for and acquire outstanding events awaiting
5 processing by that application. Update events are generically handled by
6 applications through the generation of drawing events for at least the affected
7 portion of the application window.

8
9 Whenever a drawing event is logically directed against a particular one of
10 the dependant windows W_1 - W_{10} by an application, the window memory is
11 correspondingly modified. If the modified portion of window is logically visible,
12 atomic drawing commands are issued to render the modification, in bit map form,
13 in a corresponding portion of the display data buffer.

14
15 The environment manager application, when executing in the overt
16 environment, consistently responds to update events for the window W_3 to display
17 the contents of the window W_3 on the root window W_0 . Since the contents of the
18 window W_3 exists as a bit map, the update is a bit map transfer of the affected
19 portion of the window W_3 to the display data buffer. The window W_3 therefore
20 appears in the overt environment as a conventional window, generally as
21 represented in Figure 5b.

22
23 The content of the environment manager application window W_3 is
24 determined by the applications that execute within the covert environment. When
25 the operating system 40 is executing in the covert environment, the application list

60, the tree structured window list pointer 62 and active window pointer 64 are exchange copied with the alternates 70,72,74 stored in the environment manager application 42₃. The root entry of the window list during covert environment execution therefore corresponds to the window W₃. The initial application executing in the covert environment is the environment manager application 42₃. Thus, the initial application list 70 includes only the environment manager application 42₃. The pointer in the active application storage location 72 initially points to this application.

Again referring to Figure 5a, as applications are subsequently launched and begin to execute in the covert environment, window list entries are created for the windows W₁₁-W₁₆ corresponding to the covert applications. As each covert application window is created, manipulated and destroyed, drawing and update events are generated and processed by the covert applications. The covert applications directly or indirectly call the covert display driver 82 with the result that the covert root window W₃ is correspondingly updated.

Each time execution switches from the covert to the overt environment, the content of the covert root window W₃ properly reflects the visible state of the covert environment application windows. Since a single input queue handler routine 78 executes in both overt and covert environments, thereby retaining knowledge of both input queues 56, 76, a covert environment update event can be effectively issued for the environment manager application 42₃ pending a switch to execution in the overt environment. When the overt environment update event is handled, the environment manager application 42₃ correspondingly updates the

1 appropriate visible portion of the covert root window W_3 into the overt window W_3
2 on the overt root display W_0 .

3
4 Referring now to Figure 6, a flow chart illustrating the conventional initial
5 flow of operation in an application consistent with the present invention for a
6 cooperatively multitasking operating system is shown. When an application has
7 been loaded by the operating system 40, a start event or equivalent is associated
8 with the application. Until the start event is processed by the operating system 40,
9 the application remains loaded in memory in an effectively suspended state.
10 Generally at the first available opportunity, the start event is processed by the
11 operating system and an initialization routine 100 of the application is executed.
12 The initialization routine 100 performs whatever functions are necessary to
13 initialize the application including, as appropriate, the creation of a window data
14 space and the issuance of requests to establish a predetermined set of window
15 frame decorations, such as pull-down menu lists and scroll bars. Once the
16 application has been initialized, execution passes to a read next event routine
17 102. This routine 102 executes an operating system call that waits on the
18 existence of an input event from the input queue 56 specific to this application.
19 When the operating system 40 determines that such an event exists and should
20 be processed by the application, the operating system call returns with the input
21 data and execution continues with the dispatch event handler 104. The type of
22 input event is determined by the dispatch event handler 104 and an appropriate
23 sub-routine within the application is called. These sub-routines include basic
24 operations such as redraw window 106 which are utilized to handle standard
25 update event inputs as well as other sub-routines specific to the particular function

1 of the application, as illustrated generically by the application specific event
2 routine 108. Another standard event that is dispatched by the dispatch event
3 handler 104 is a terminate event, which is processed by a stop routine 110.
4

5 The corresponding operation of the present invention for cooperatively
6 mutlitasking operating systems is generally shown in Figure 7. Upon start of the
7 environment manager application, the initialization routine 120 is executed. The
8 initializations performed include the following steps:
9

10 1. Request allocation of memory space for an alternate input queue
11 structure. Initialize the structure to a clear or empty state.
12

13 2. Request allocation of memory space for a pointer to an alternate
14 tree structured window list.
15

16 3. Request allocation of an alternate current active window pointer
17 storage location.
18

19 4. Request allocation of an alternate applications list structure.
20 Initialize the application list structure to a clear or empty state. Insert an initial
21 entry into the application list referencing the Environment Manager application.
22 Insert a pointer to this application in the alternate current active window storage
23 location.
24

1 5. Execute an operating system call to allocate memory for a covert
2 environment window. Initialize the memory space to represent a predefined
3 background pattern. Request allocation of memory space for a root window list
4 entry and establish a pointer stored in the alternate tree structured window list
5 pointer storage location. Initialize the root window list entry.
6

7 6. Execute operating system calls to establish window frame
8 decorations for an overt environment window.
9

10 7. Execute operating systems calls to add menu items to the overt
11 environment window frame to establish links to executable features of the
12 environment manager, including an application selector and launcher for
13 selecting and launching an application in the covert environment (desktop rules
14 may also be implemented at the system configuration level to support the drag
15 and drop of application icons onto the environment manager window to launch
16 the application).
17

18 8. Replace the input handler routine with the alternate handler routine
19 provided as part of the environment manager application.
20

21 9. Execute an operating system call to establish an operating system
22 alarm for periodically generating a timer event directed to the environment
23 manager application.
24

1 Execution then continues with the execution of the read event from overt
2 queue step 122. This results in the execution of an operating system call that
3 waits on the existence of an input event in the input queue destined for the
4 environment manager application. The timer event established as part of the
5 initialization routine 120 guarantees that a periodic event will exist for the
6 environment manager application.
7

8 When an event exists for the environment manager application, execution
9 continues with the dispatch to the overt event handler routine 124. The expected
10 events, in a preferred embodiment of the present invention, include timer events,
11 menu selection events, character input events, a termination event and other
12 events related to the manipulation of the window frame and decorations. A
13 combination of menu choice and character input events may be utilized to select
14 an application for launching within the covert environment. When an application
15 is launched in this or an equivalent manner, the application is loaded under the
16 control of the environment manager application into the memory space of the
17 operating system 40 in a suspended state. That is, upon loading the application
18 is added to the application list for the covert rather than overt application list.
19 Consequently, the application will be initialized and begin execution exclusively
20 within the covert environment. Preferably, the operating system top of memory
21 value is used in defining the loading address of the application. Further, this
22 value is preferably stored separate from the application lists and is not modified
23 merely as a consequence of a switch between the overt and covert environments.
24

1 A terminate event is also handled by the dispatch to overt handler routine
2 124. The terminate event causes execution of a terminate routine that provides
3 for the restoration of the input queue handler routine 58. A final copy exchange,
4 as needed, of the various data structures and routines is then performed. Finally,
5 the resources allocated to the environment manager application and any covert
6 applications upon initialization or through execution are released back to the
7 operating system 40.

8
9 When the environment manager application receives a non-terminal overt
10 event from overt queue, execution passes to a decision point routine 126 to
11 determine whether to switch between the overt and covert environments. This
12 decision is made based upon the relative number of pending events in the overt
13 and covert input queues 56, 76. Where the overt queue 56 has the larger
14 number of pending events, execution continues with the read event from overt
15 queue routine 122.

16
17 However, if the covert queue 76 has the larger number of pending events,
18 execution continues with a switch to covert environment routine 128. This routine
19 128 provides for the copy exchange of the application lists 60, 70, pointers to the
20 tree structure window lists 62, 72, and active window pointers 64, 74. The
21 contents of the window queues 56, 76 are also copy exchanged. Finally, the
22 drawing jump tables 66, 80 are copy exchanged to complete the switch to the
23 covert environment. Execution then continues with a read event from covert queue
24 routine 130.

25

1 In executing within the covert environment, a very small number of
2 potential events will be directed to the environment manager. All events
3 associated with the window frame decorations and menus will be generated within
4 the overt environment. Focus events exist for the windows of applications
5 executing within the covert environment. However, the window of the environment
6 manager application is, in effect, the root display of the covert environment.
7 Thus, no covert environment focus events directed to the environment manager
8 are expected. However, timer events continue to occur. The request for periodic
9 timer events made during the execution of the initialization routines 120 remains
10 with the operating system 40 independent of whether the operating system is
11 executing in the overt or covert environments.

12
13 Execution then passes with the event obtained by the read event from covert
14 queue routine 130 to the dispatch to covert handler routine 132. Once an
15 appropriate handler routine has been called and executed, execution then
16 continues to a decision point routine 134. As with the routine 126, the decision
17 point routine 134 determines which input queue has the greater number of
18 pending events. If the covert queue 76 has the greater number of pending events,
19 execution continues with the read event from the covert queue routine 130.
20 Conversely, if the input queue 56 has the greater number of pending events,
21 execution continues with a switch to overt environment routine 136.

22
23 The switch to overt environment routine 136 performs substantially the
24 same function as the switch to covert environment routine 128. The various data
25 structures are copy exchanged between the operating system 40 and environment

1 manager application 42₃. The switch to overt environment routine 136 may enter
2 an update event into the overt input queue 56 for the window corresponding to
3 the environment manager application. Whether this update event is entered
4 depends on whether the covert environment modified any visible portion of the
5 covert root display window W₃. This update event will typically be the next event
6 read from the overt queue by the read event from overt queue routine 122. When
7 read, the event will invoke an update of the environment manager application
8 window to the root window of the overt environment. Thus, updates made to the
9 covert root window due to the execution of applications in the covert environment
10 will be properly reflected in the environment manager application window W₃
11 upon or shortly after a switch is made back to the overt environment
12

13 For the case of a preemptively multi-tasking operating system, or at least
14 where the operating system provides a task switch notification event, the operation
15 of the present invention may be substantially simplified. Rather than requiring the
16 environment manager application to be scheduled and executed as an ordinary
17 application, the environment switching routines provided by the environment
18 manager application can be effectively registered with the operating system and
19 invoked automatically by the operating system in response to each task switch
20 implemented by the operating system. In this case, the initializations described
21 above are performed as part of the initialization of the environment manager
22 application. However, a second applications list is not required. All applications
23 remain visible at all times to the operating system. However, the environment
24 manager tracks which applications are in the overt and covert environments
25 based on the environment in which the application is launched. Specifically,

applications launched into the covert environment are launched with the support of the environment manager and can thus be identified as covert environment applications. All other applications are overt environment applications. Thus, upon a task switch to a new task, the environment switching routines of the environment manager are executed. If the application of the task that is the target of the task switch is in a different environment, then data structures are exchange copied as appropriate to switch to the overt or covert environment. No decision as to which environment has the largest number of pending input events, since all input events result in the flagging of corresponding applications as ready to run. By the use of the single application list, the scheduler will properly select applications from both environments to run in a prioritized order.

A flow diagram illustrating the function of a standard input handler routine 58 is shown in Figure 8. As illustrated, an operating system input event handler sub-routine 140 is responsible for performing the interrupt handling or other low level functions necessary to acquire input data provided via a keyboard or mouse 44 or from a communications driver 50 coupled to the operating system 40 as represented by the logical data path 96. This input data is then passed to an add input event to queue routine 142. In the case of simple character input, the intended destination of the input event is the currently active window, determinable from the active window pointer 64. In general for a mouse event, the operating system input focus is changed to the top most window under the mouse pointer at the point of the mouse event. The active window pointer 64 is updated appropriately. Concurrently, an update window event and any other events bound to the mouse event may be created. Each of these events are added

1 to the input queue 56 with the logical destination identified as being the current
2 active window. Execution is then returned to the operating system at 144 as
3 appropriate to allow resumption of the execution of the interrupted application.
4

5 In accordance with the present invention, a new add input event to queue
6 input handler 142 is provided as the handler routine 78 to enable management
7 of both the overt and covert event input queues 56, 76. As shown in Figure 9, an
8 initial decision point determination is made 150 as to whether a new input event
9 is associated with the overt or covert environment. The current active
10 environment, overt or covert, is maintained in a global state variable at all times
11 by the environment manager application. Thus, the active window in the overt
12 environment can be correctly identified from one of the active window pointer
13 storage locations 64,74, regardless of whether the overt or covert environment is
14 active at the time of the input event.
15

16 If the current active window in the overt environment is not the environment
17 manager application, the input event is added to the overt input queue 56 at 152
18 and execution is returned to the operating system at 154. In a preferred
19 embodiment, the event is added to the overt queue 56 under control of the input
20 handler routine 78 and specifically through execution of the handler routine 58.
21 That is, the input handler routine 78 executes to swap in, as necessary, the various
22 data structures appropriate for the overt environment. The input handler 58 is
23 then called to insert the event into the overt queue 56. The state of the
24 corresponding application may also be marked as ready to run in the overt
25 application list. The input handler routine 78 then resumes execution to swap

1 back the various data structures as appropriate to revert to the environment
2 existing when the routine 78 was first called.

3
4 If the overt environment active window for the input event is the window of
5 the environment manager application, a further determination is made as to
6 whether the input event is a mouse event at 156. If the event is not a mouse
7 event, the input data is added to the covert input queue 76 at 158. Again, in a
8 preferred embodiment, the event is added to the covert queue 76 by swapping,
9 if needed, to the covert environment state, executing the input handler routine 58
10 to insert the event and set a corresponding application as ready to run, and
11 swapping back, if needed, to the environment existing when the routine 78 was
12 called. Execution then returns to the operating system at 154.

13
14 Where the mouse event occurs with the mouse pointer within the window
15 of the environment manager application, both input focus and the designation of
16 the current active window in the overt environment remains or is set to the
17 environment manager application. The mouse event must, however, be further
18 localized within the covert environment root display space to associate the mouse
19 event with a potential change of input focus and at least a cursor position change
20 within the covert environment. Accordingly, a translation routine 162 is executed
21 to convert or map the location of the event within the environment manager
22 application window to the covert root window coordinate system. Thus, the mouse
23 pointer location is translated into view window relative coordinates before the
24 event, including the location of the event representing the logical destination of
25 the event, is added to the covert input queue 76 in the same manner as other

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

3
4
5
6
7

8
9
10
11
12
13
14
15
16

17
18
19
20
21
22
23
24
25

1 input queue could be used, though the operating system routines that determine
2 the application for which an event is destined would need to be modified to
3 account for the difference in the overt and covert environment coordinate systems.
4 A single window tree structure could also be used, though again the system
5 routines for walking the tree structure would need to be modified to distinguish
6 between overt and covert environment portions. The covert device driver could
7 be included as a standard part of the operating system or system drivers, since the
8 driver is essentially device independent. Alternately, the function of the covert
9 environment device driver could be modified to selectively operate in line with the
10 conventional display driver. Atomic drawing commands from covert environment
11 applications can be passed to the covert device driver, suitably modified to be
12 covert window relative, and then passed via the jump table 66 to the conventional
13 display device driver. Finally, switching between the overt and covert
14 environments could be provided for as part of the normal operation of the
15 scheduler.

16
17 Thus, a system providing for the establishment of two or more environment
18 spaces for the execution of applications within a common operating system has
19 been described. The alternate execution environments provide a convenient and
20 reliable environment for the execution of applications that may be share among
21 collaborative hosts. The resulting collaborative sessions, by isolation in alternate
22 execution environments, facilitate the execution of both private and shared
23 applications on each collaborative host in a reliable and predictable manner.